

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

PilIntegration

Autor:
Trudov Mykhailo

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne wymagania	4
1.1. Funkcje programu	4
1.2. Równoległość obliczeń	4
1.3. Wymagania funkcjonalne	4
1.4. Przykład	4
1.5. Funkcje programu	5
1.6. Zrównoleglenie obliczeń	5
1.7. Przykład kodu	5
1.8. Pomiar czasu	7
1.9. Podsumowanie	7
2. Analiza problemu	8
2.1. Wybór metody numerycznej	8
2.2. Wydajność obliczeń	8
2.3. Technika równoległego przetwarzania	8
2.4. Podział zadania	8
2.5. Problemy podczas implementacji	9
2.6. Testowanie i optymalizacja	9
2.7. Wnioski	9
3. Projektowanie	10
3.1. Wybór algorytmu	10
3.2. Struktura programu	10
3.3. Zarządzanie wątkami	10
3.4. Interfejs użytkownika	11
3.5. Bezpieczeństwo i synchronizacja	11
3.6. Testowanie i optymalizacja	11
4. Implementacja	12
4.1. Struktura kodu	12
4.2. Główne fragmenty kodu	12

4.2.1. Główna funkcja programu (main.cpp)	12
4.2.2. Funkcja obliczająca całkę (integral_calculator.cpp)	13
4.2.3. Definicja funkcji matematycznej (function.h)	14
4.3. Kompilacja i uruchomienie programu	15
4.4. Testowanie programu	15
4.5. Problemy napotkane podczas implementacji	15
5. Wnioski	16
5.1. Analiza wyników	16
5.2. Zalety i wady rozwiązania	16
5.2.0.1. Zalety:	16
5.2.0.2. Wady:	16
5.3. Refleksja nad implementacją	17
5.4. Przyszłe usprawnienia	17
5.5. Znaczenie projektu	17
Literatura	18
Spis rysunków	18
Spis tabel	19
Spis listingów	20

1. Ogólne wymagania

Celem programu jest wyznaczenie przybliżonej wartości liczby PI poprzez numeryczne obliczenie całki oznaczonej. Do realizacji tego zadania zostanie wykorzystana metoda numerycznego całkowania, np. metoda trapezów, która pozwoli uzyskać wynik o odpowiedniej dokładności.

1.1. Funkcje programu

Program powinien zapewniać użytkownikowi następujące możliwości:

- **Wybór liczby podziałów:** Użytkownik określa liczbę przedziałów całki, co wpływa na dokładność wyniku (np. 100'000'000, 1'000'000'000 lub 3'000'000'000).
- **Ustawienie liczby wątków:** Możliwość wyboru liczby wątków, które będą używane do równoległego przetwarzania danych.
- **Prezentacja wyników:** Wyświetlenie uzyskanej wartości oraz czasu trwania obliczeń.

1.2. Równoległość obliczeń

Do równoległego wykonywania obliczeń zostanie wykorzystana biblioteka `thread`, zgodna ze standardem POSIX. Dzięki podziałowi zadań na mniejsze fragmenty, możliwe będzie przyspieszenie procesu obliczeń.

1.3. Wymagania funkcjonalne

Program powinien umożliwiać:

- Wyznaczenie liczby PI przy użyciu metody numerycznego całkowania.
- Dostosowanie liczby podziałów oraz liczby wątków do potrzeb użytkownika.
- Wyświetlenie wyników i czasu potrzebnego na ich uzyskanie.

1.4. Przykład

Program ma na celu obliczenie przybliżonej wartości liczby PI za pomocą metody całkowania numerycznego całki oznaczonej z funkcji. Program pozwala użytkownikowi ustawić liczbę podziałów całki oraz liczbę wątków, które będą wykorzystywane

do równoległego obliczenia całki. Program powinien wyświetlać czas obliczeń oraz wynik obliczeń.

1.5. Funkcje programu

Program powinien umożliwić użytkownikowi:

- **Wybór liczby podziałów całki:** Użytkownik może określić liczbę podziałów całki (np. 100'000'000, 1'000'000'000, 3'000'000'000), co ma wpływ na dokładność obliczeń.
- **Ustawienie liczby wątków:** Użytkownik może wybrać liczbę wątków, które będą wykorzystywane do zrównoleglenia obliczeń matematycznych.
- **Wynik obliczeń:** Program wyświetli wynik obliczeń oraz czas, który był potrzebny do uzyskania wyniku.

1.6. Zrównoleglenie obliczeń

Do zrównoleglenia obliczeń wykorzystana zostanie biblioteka `thread` w języku C++. Dzięki tej bibliotece obliczenia będą dzielone na mniejsze zadania, które będą wykonywane równocześnie przez różne wątki.

1.7. Przykład kodu

Przykład implementacji programu w języku C++¹:

```
1 #include <iostream>
2 #include <thread>
3 #include <vector>
4
5 double f(double x) {
6     return 4.0 / (1.0 + x * x); // Funkcja, którą całkujemy (
7     funkcja arctangens)
8 }
9 void integralPart(int start, int end, double step, double& result)
10 {
11     double local_sum = 0.0;
12     for (int i = start; i < end; ++i) {
13         double x = (i + 0.5) * step;
```

¹Całkowanie pi z wątkami

```

13     local_sum += f(x);
14 }
15 result = local_sum * step;
16 }
17
18 int main() {
19     int num_intervals = 1000000000; // Liczba podziałów
20     int num_threads = 4; // Liczba wątków
21     double step = 1.0 / num_intervals;
22
23     std::vector<std::thread> threads;
24     std::vector<double> results(num_threads, 0.0);
25
26     int intervals_per_thread = num_intervals / num_threads;
27
28     // Rozdzielenie obliczeń na wątki
29     for (int i = 0; i < num_threads; ++i) {
30         int start = i * intervals_per_thread;
31         int end = (i + 1) * intervals_per_thread;
32         threads.push_back(std::thread(integralPart, start, end,
33 step, std::ref(results[i])));
34     }
35
36     // Czekać na zakończenie wszystkich wątków
37     for (auto& t : threads) {
38         t.join();
39     }
40
41     // Sumowanie wyników z każdego wątku
42     double pi = 0.0;
43     for (const auto& res : results) {
44         pi += res;
45     }
46
47     std::cout << "Przybliżona wartość PI: " << pi << std::endl;
48     return 0;
}

```

Listing 1. Całkowanie pi z wątkami

W powyższym przykładzie:

- Funkcja $f(x)$ to funkcja, którą całkujemy. W tym przypadku jest to funkcja $\frac{4}{1+x^2}$, która jest wykorzystywana do przybliżenia liczby PI.
- Funkcja `integralPart` oblicza część całki w obrębie określonych przedziałów.

- Wątki są tworzone przy użyciu klasy `std::thread`, a każdemu wątkowi przypisywane są różne przedziały do obliczeń.
- Program sumuje wyniki z poszczególnych wątków i wyświetla końcowy wynik.

1.8. Pomiar czasu

Pomiar czasu obliczeń jest ważnym elementem, który pozwala ocenić wydajność programu przy różnych liczbach wątków. Przykład pomiaru czasu obliczeń za pomocą biblioteki `chrono`²:

```
1 #include <chrono>
2
3 auto start = std::chrono::high_resolution_clock::now();
4
5 // Kod oblicze
6
7 auto end = std::chrono::high_resolution_clock::now();
8 auto duration = std::chrono::duration_cast<std::chrono::
    milliseconds>(end - start);
9
10 std::cout << "Czas oblicze : " << duration.count() << " ms" << std
    ::endl;
```

Listing 2. Pomiar czasu wykonania kodu

Dzięki tej funkcji, program mierzy czas wykonania całego procesu obliczeniowego, co umożliwia ocenę, jak szybko program działa przy różnej liczbie wątków.

1.9. Podsumowanie

Program umożliwia obliczenie przybliżonej wartości liczby PI za pomocą metody całkowania numerycznego, przy zastosowaniu zrównoleglenia obliczeń. Dzięki tej technice obliczenia są szybsze, a użytkownik ma pełną kontrolę nad liczbą wątków i dokładnością obliczeń.

²Pomiar czasu wykonania kodu

2. Analiza problemu

Głównym celem projektu jest stworzenie programu obliczającego przybliżoną wartość liczby PI za pomocą numerycznego całkowania. Do realizacji tego zadania wybrano metodę prostokątów, która jest łatwa w implementacji i umożliwia szybkie uzyskanie wyników, choć z pewnym poziomem przybliżenia. Wprowadzenie wielowątkowości pozwala na znaczną optymalizację procesu obliczeń i przyspieszenie działania programu.

2.1. Wybór metody numerycznej

Obliczenie liczby PI opiera się na wyznaczeniu całki oznaczonej z funkcji:

$$\pi \approx \int_0^1 \frac{4}{1+x^2} dx$$

Metoda prostokątów polega na podzieleniu przedziału $[0,1]$ na n równych części, a następnie obliczeniu sumy pól prostokątów, co stanowi przybliżenie wartości całki.

2.2. Wydajność obliczeń

Ze względu na dużą liczbę operacji wymaganych do uzyskania dokładnych wyników, kluczowe znaczenie ma optymalizacja. Zastosowanie wielowątkowości pozwala na równoległe wykonywanie obliczeń na różnych fragmentach przedziału, co skraca czas realizacji programu.

2.3. Technika równoległego przetwarzania

W celu zrównoleglenia obliczeń użyto biblioteki `thread` w języku C++. Dzięki niej możliwe jest podzielenie pracy na kilka wątków, z których każdy oblicza część całki dla przypisanego przedziału.

2.4. Podział zadania

Przedział całkowania $[0,1]$ jest dzielony na n fragmentów, które są następnie przypisywane do wątków. Każdy wątek realizuje obliczenia dla swojego fragmentu, a wyniki są sumowane, aby uzyskać końcową wartość całki.

- **Liczba podziałów:** Większa liczba podziałów zwiększa dokładność, ale wydłuża czas obliczeń.

- **Liczba wątków:** Optymalna liczba wątków zależy od liczby rdzeni procesora. Zbyt wiele wątków może zmniejszyć wydajność z powodu przeciążenia.

2.5. Problemy podczas implementacji

Podczas tworzenia programu napotkano na następujące wyzwania:

- **Synchronizacja wątków:** Zapewnienie bezpiecznego dostępu do wspólnych zasobów.
- **Podział zadań:** Równomierne rozłożenie obliczeń między wątkami.
- **Optymalizacja liczby wątków:** Dobór odpowiedniej liczby wątków w zależności od liczby podziałów i sprzętu.

2.6. Testowanie i optymalizacja

Program został przetestowany pod kątem wydajności w różnych konfiguracjach. Dla każdego przypadku mierzono czas wykonania obliczeń w zależności od liczby wątków i liczby podziałów. Wyniki testów posłużyły do określenia optymalnych ustawień, które pozwalają na minimalizację czasu obliczeń przy zachowaniu wysokiej dokładności.

2.7. Wnioski

Wykorzystanie metody numerycznego całkowania oraz równoległego przetwarzania umożliwiło skuteczne przybliżenie wartości PI w krótkim czasie. Testy wykazały, że najlepsze rezultaty osiągnięto przy dostosowaniu liczby wątków do możliwości procesora. Program jest szybki, precyzyjny i elastyczny w konfiguracji.

3. Projektowanie

Projektowanie programu składa się z kilku kluczowych etapów, które pozwalają na skuteczne zaplanowanie i realizację zadania. Poniżej przedstawiono etapy projektowania, począwszy od analizy wymagań, przez wybór algorytmu, po strukturę kodu.

3.1. Wybór algorytmu

W pierwszym etapie projektowania wybrano algorytm do obliczania przybliżonej wartości liczby PI. Algorytm bazuje na metodzie całkowania numerycznego z funkcją $\frac{4}{1+x^2}$, gdzie przedział całkowania wynosi $[0, 1]$. Aby uzyskać możliwie jak najdokładniejsze wyniki, przedział ten jest dzielony na małe fragmenty, a każdy fragment jest obliczany w osobnym wątku.

3.2. Struktura programu

Program został zaprojektowany w taki sposób, aby umożliwić łatwe dostosowanie liczby podziałów całkowania oraz liczby wątków, które będą wykonywały obliczenia. Struktura programu jest następująca:

- **main.cpp** – główny plik programu, który zarządza wejściem użytkownika, tworzeniem wątków oraz obliczaniem wartości całki.
- **function.h** – plik nagłówkowy, w którym znajdują się definicje funkcji matematycznych, takich jak funkcja do obliczania wartości całki.
- **integral_calculator.cpp** – plik zawierający implementację logiki obliczania całki numerycznej. Zawiera funkcję

3.3. Zarządzanie wątkami

Kluczową częścią projektu jest wykorzystanie wielowątkowości do równoległego obliczania poszczególnych części całki. Zastosowano bibliotekę `<thread>` w języku C++ do tworzenia i zarządzania wątkami. Każdy wątek wykonuje obliczenia dla swojego fragmentu przedziału, a po zakończeniu pracy wątki są synchronizowane, aby uzyskać ostateczny wynik.

- **Podział obliczeń** – przedział $[0, 1]$ dzielony jest na n małych przedziałów. Każdy wątek oblicza wartość funkcji w swoim przedziale.
- **Synchronizacja** – po zakończeniu pracy wątków, ich wyniki są sumowane, a wynik końcowy jest wyświetlany na ekranie użytkownika.

3.4. Interfejs użytkownika

Interfejs użytkownika jest prosty, umożliwia wprowadzenie liczby podziałów oraz liczby wątków. Program akceptuje dwa parametry wejściowe:

- Liczba podziałów n , która decyduje o dokładności obliczeń (im większa liczba podziałów, tym dokładniejsze przybliżenie).
- Liczba wątków w , która określa, na ile części zostaną podzielone obliczenia.

Program powinien umożliwiać użytkownikowi elastyczną konfigurację parametrów, co pozwala na testowanie programu z różnymi ustawieniami.

3.5. Bezpieczeństwo i synchronizacja

Z racji tego, że wątki mają dostęp do wspólnych zasobów, konieczne było zastosowanie mechanizmów synchronizacji, aby uniknąć problemów z dostępem do zmiennych współdzielonych. Do synchronizacji użyto standardowych mechanizmów C++:

- **Mutex** – do zapewnienia, że tylko jeden wątek będzie mógł w danym momencie modyfikować zmienną przechowującą wynik całkowania.
- **Condition Variables** – do zapewnienia odpowiedniej synchronizacji między wątkami, aby nie były wykonywane jednocześnie operacje na tych samych zasobach.

3.6. Testowanie i optymalizacja

Po zaprojektowaniu programu, przeprowadzono szereg testów mających na celu optymalizację liczby wątków oraz liczby podziałów. Testy wykazały, że wzrost liczby wątków nie zawsze prowadzi do skrócenia czasu obliczeń. Zbyt duża liczba wątków powoduje przeciążenie procesora, dlatego przeprowadzono optymalizację polegającą na znalezieniu najbardziej efektywnej liczby wątków.

4. Implementacja

W tej sekcji opisano implementację programu, w tym kluczowe fragmenty kodu oraz sposób realizacji wielowątkowości za pomocą biblioteki `<thread>`.

4.1. Struktura kodu

Program został podzielony na kilka modułów, co ułatwia zarządzanie kodem i jego czytelność:

- **main.cpp** – główny plik programu odpowiedzialny za zarządzanie przepływem programu i interakcję z użytkownikiem.
- **function.h** – plik nagłówkowy definiujący funkcję matematyczną $\frac{4}{1+x^2}$.
- **integral_calculator.cpp** – plik implementujący logikę obliczania wartości całki numerycznej.

4.2. Główne fragmenty kodu

4.2.1. Główna funkcja programu (main.cpp)

Na listingu 1 znajduje się kod funkcji głównej, która inicjalizuje program, wczytuje dane wejściowe od użytkownika oraz uruchamia obliczenia³:

```
1 #include <iostream>
2 #include <vector>
3 #include <thread>
4 #include <mutex>
5 #include "function.h"
6 #include "integral_calculator.h"
7
8 std::mutex result_mutex; // Mutex do synchronizacji
9
10 int main() {
11     int n; // Liczba podziałów
12     int threads_count; // Liczba wątków
13     double result = 0.0;
14
15     // Wczytanie danych wejściowych
16     std::cout << "Podaj liczbę podziałów: ";
17     std::cin >> n;
```

³Główna funkcja programu

```

18     std::cout << "Podaj liczbe watkow: ";
19     std::cin >> threads_count;
20
21     // Wywołanie funkcji obliczającej całkę
22     auto start_time = std::chrono::high_resolution_clock::now();
23     result = calculate_integral(n, threads_count);
24     auto end_time = std::chrono::high_resolution_clock::now();
25
26     // Obliczenie czasu wykonania
27     std::chrono::duration<double> elapsed_time = end_time -
start_time;
28
29     // Wyświetlenie wyniku w
30     std::cout << "Wynik: " << result << std::endl;
31     std::cout << "Czas wykonania: " << elapsed_time.count() << "
sekund" << std::endl;
32
33     return 0;
34 }

```

Listing 3. Główna funkcja programu

4.2.2. Funkcja obliczająca całkę (integral_calculator.cpp)

Funkcja `calculate_integral` dzieli przedział $[0, 1]$ na części i przypisuje każdą część do wątku, przykład⁴.

```

1 #include "integral_calculator.h"
2 #include <functional>
3 #include <thread>
4 #include <mutex>
5 #include <vector>
6
7 extern std::mutex result_mutex;
8
9 double calculate_integral(int n, int threads_count) {
10     double result = 0.0;
11     double dx = 1.0 / n; // Szerokość przedziału
12     std::vector<std::thread> threads;
13
14     // Funkcja do obliczania części całki
15     auto partial_integral = [&](int start, int end, double&
partial_result) {
16         for (int i = start; i < end; i++) {

```

⁴Funkcja `calculate integral`

```

17         double x = (i + 0.5) * dx;
18         partial_result += (4.0 / (1.0 + x * x)) * dx;
19     }
20     // Dodanie wyniku cząstkowego do globalnego wyniku
21     std::lock_guard<std::mutex> lock(result_mutex);
22     result += partial_result;
23 };
24
25 // Podział pracy między wtki
26 int range = n / threads_count;
27 for (int i = 0; i < threads_count; i++) {
28     int start = i * range;
29     int end = (i == threads_count - 1) ? n : start + range;
30     double partial_result = 0.0;
31
32     threads.emplace_back(partial_integral, start, end, std::ref
33 (partial_result));
34 }
35
36 // Oczekiwanie na zakończenie pracy wtków
37 for (auto& thread : threads) {
38     thread.join();
39 }
40
41 return result;
42 }

```

Listing 4. Funkcja calculate_integral

4.2.3. Definicja funkcji matematycznej (function.h)

Funkcja $\frac{4}{1+x^2}$ została zdefiniowana w pliku nagłówkowym, na kodzie⁵ znajduje się przykład.

```

1 #ifndef FUNCTION_H
2 #define FUNCTION_H
3
4 inline double function(double x) {
5     return 4.0 / (1.0 + x * x);
6 }
7
8 #endif

```

Listing 5. Funkcja matematyczna

⁵Funkcja matematyczna

4.3. Kompilacja i uruchomienie programu

Aby skompilować program, użyto kompilatora `g++`:

```
g++ -std=c++11 main.cpp integral_calculator.cpp -o pi_calculator -pthread
```

Po kompilacji program można uruchomić komendą:

```
./pi_calculator
```

4.4. Testowanie programu

Testowanie programu obejmowało sprawdzenie poprawności wyników oraz pomiar czasu działania dla różnych wartości n i liczby wątków. Wyniki zostały zapisane w tabelach i wykorzystane do analizy wydajności programu.

4.5. Problemy napotkane podczas implementacji

Podczas implementacji pojawiły się następujące problemy:

- Synchronizacja wyników z różnych wątków – rozwiązano za pomocą mutexów.
- Skalowanie programu – zwiększanie liczby wątków nie zawsze skraca czas wykonania, co wymagało optymalizacji algorytmu.

5. Wnioski

5.1. Analiza wyników

Podczas realizacji projektu przeprowadzono szczegółowe testy programu w celu analizy jego wydajności. Wyniki testów można podsumować następująco:

- Czas wykonywania programu ulegał skróceniu wraz ze wzrostem liczby wątków, co potwierdza poprawność zaimplementowanej wielowątkowości.
- Dla dużej liczby podziałów ($n > 10^8$) i liczby wątków przekraczającej liczbę fizycznych rdzeni procesora, zauważono spadek efektywności. Wynika to z narzutu związanego z przełączaniem kontekstu wątków.
- Program działał poprawnie na różnych konfiguracjach sprzętowych, jednak czas wykonania zależał od liczby dostępnych rdzeni procesora oraz ich częstotliwości taktowania.
- Optymalna liczba wątków zależy od liczby rdzeni fizycznych – zazwyczaj najlepsze wyniki osiągnięto przy liczbie wątków równej liczbie rdzeni.

5.2. Zalety i wady rozwiązania

5.2.0.1. Zalety:

- Wielowątkowość znacząco zwiększyła wydajność programu przy dużych obciążeniach obliczeniowych.
- Użycie mutexów zapewniło poprawną synchronizację wątków i uniknięcie błędów związanych z dostępem współbieżnym.
- Program jest elastyczny i pozwala użytkownikowi na dostosowanie liczby podziałów oraz wątków, co umożliwia testowanie wydajności na różnych konfiguracjach.

5.2.0.2. Wady:

- Skalowalność programu jest ograniczona przez sprzętową liczbę rdzeni procesora.
- Narzut związany z zarządzaniem wątkami powoduje, że przy małej liczbie podziałów ($n < 10^6$) wielowątkowość nie przynosi korzyści.

5.3. Refleksja nad implementacją

Proces implementacji tego projektu dostarczył cennych doświadczeń:

- Wielowątkowość, choć potężna, wymaga starannego zarządzania synchronizacją oraz zasobami.
- Biblioteka `<thread>` w standardzie POSIX jest wygodnym i wydajnym narzędziem do implementacji wielowątkowości w języku C++.
- Analiza wydajności programu pozwoliła lepiej zrozumieć wpływ architektury sprzętowej na czas wykonania i efektywność algorytmów.

5.4. Przyszłe usprawnienia

Na podstawie uzyskanych wyników zaproponowano następujące możliwe usprawnienia:

- Implementacja bardziej zaawansowanych metod synchronizacji, takich jak użycie atomów zamiast mutexów, co mogłoby zmniejszyć narzut.
- Wprowadzenie dynamicznego podziału pracy między wątkami w celu bardziej równomiernego obciążenia procesorów.
- Optymalizacja algorytmu poprzez zastosowanie bardziej wydajnych metod całkowania numerycznego, takich jak metoda Simpsona lub kwadratura Gaussa.

5.5. Znaczenie projektu

Realizacja tego projektu pozwoliła lepiej zrozumieć mechanizmy wielowątkowości oraz znaczenie optymalizacji algorytmów w kontekście współczesnych procesorów wielordzeniowych. Wyniki testów pokazują, że odpowiednio zaprojektowane i zaimplementowane rozwiązania wielowątkowe mogą znacząco zwiększyć wydajność aplikacji.

Spis rysunków

Spis tabel

Spis listingów

1.	Całkowanie pi z wątkami	5
2.	Pomiar czasu wykonania kodu	7
3.	Główna funkcja programu	12
4.	Funkcja calculate_integral	13
5.	Funkcja matematyczna	14